Chapters and Articles

You might find these chapters and articles relevant to this topic.

Chapter

[Implementation Using Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9781856179218000249)

20.3 Floating Point

Nearly all DSP processors are fixed-point processors. This is due to the larger floating-point circuit requirements, higher power consumption, and lower performance compared to [fixed point](https://www.sciencedirect.com/topics/engineering/fixed-points). For the vast majority of DSP applications, fixed-point arithmetic is suitable. However, this does require care on the part of the designer to ensure the dynamic range of the signal is mapped into the limited fixed-point precision.

Several DSP processors do offer floating-point DSP products, although at much lower performance levels than the fixed-point products. One popular application for floating-point DSP processors is high-fidelity [audio processing](https://www.sciencedirect.com/topics/computer-science/audio-signal-processing). This is due to relatively low processing rates, high dynamic range requirements, and extensive use of [IIR filters](https://www.sciencedirect.com/topics/engineering/iir-filter), which can have stability issues when implemented in fixed point.

High-performance floating point is more commonly implemented on Pentium-type processors, [graphics processors](https://www.sciencedirect.com/topics/computer-science/graphics-processor), or some specialty floating-point processors. Usage tends to be on military applications, such as radar back-end processing, or high-performance computing, such as for research purposes (an example might be climate simulations).

Book

[2010, Digital Signal Processing 101](https://www.sciencedirect.com/book/9781856179218/digital-signal-processing-101)

Michael Parker

Chapter

[Implementation Using Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9780128114537000275)

27.3 Floating Point

Most [DSP](https://www.sciencedirect.com/topics/computer-science/digital-signal-processing) processors are [fixed point](https://www.sciencedirect.com/topics/computer-science/fixed-points) processors. This is due to the larger floating point circuit requirements, higher power consumption and lower performance compared to fixed point. For the vast majority of [DSP applications](https://www.sciencedirect.com/topics/computer-science/signal-processing-application), [fixed point arithmetic](https://www.sciencedirect.com/topics/engineering/fixed-point-arithmetic) is suitable. However, this does require care on part of designer to ensure dynamic range of the signal is mapped into the limited fixed point precision.

Several DSP processors do offer floating point DSPs products, initially at much lower performance levels than the fixed point products. One popular application for floating point DSP processors is high fidelity [audio processing](https://www.sciencedirect.com/topics/computer-science/audio-signal-processing). This is due to relatively low processing rates, high dynamic range requirements, and extensive use of [infinite impulse response filters](https://www.sciencedirect.com/topics/computer-science/infinite-impulse-response-filter), which can have stability issues when implemented in fixed point.

Some more recent DSP processors offer the ability to do either 16-bit fixed point or single precision floating point at fairly high rates. The peak floating point performance of DSPs can exceed 100 GFLOPS.

However, high-performance floating point is more commonly implemented on Pentium-type processors, [graphics processors](https://www.sciencedirect.com/topics/computer-science/graphics-processor), [field programmable gate arrays](https://www.sciencedirect.com/topics/engineering/field-programmable-gate-arrays), or some specialty floating point processors. Performance levels in the multiple TFLOPS are available in some devices. Usage tends to be on military applications, such as radar back-end processing, or [high performance computing](https://www.sciencedirect.com/topics/computer-science/high-performance-computing) or [machine learning](https://www.sciencedirect.com/topics/computer-science/machine-learning). Some examples requiring high rates of floating point processing are financial modeling, [options pricing](https://www.sciencedirect.com/topics/computer-science/option-pricing), climate simulations, seismic research, and genomics.

Book

[2017, Digital Signal Processing 101 (Second Edition)](https://www.sciencedirect.com/book/9780128114537/digital-signal-processing-101)

Michael Parker

Chapter

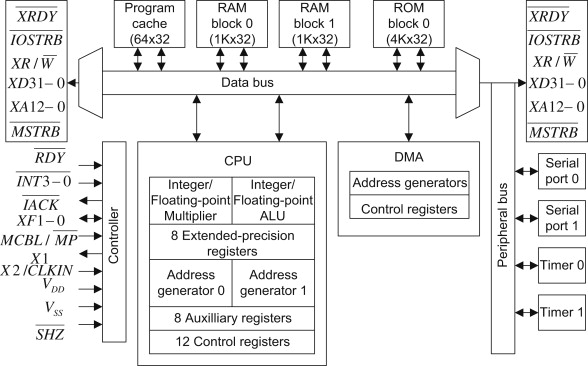
[Hardware and Software for Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9780124158931000093)

9.4.6 Floating-Point Processors

Floating-point DS processors perform DSP operations using floating-point arithmetic, as we discussed before. The advantages of using the floating-point processor include getting rid of finite word length effects such as overflows, round-off errors, truncation errors, and coefficient [quantization error](https://www.sciencedirect.com/topics/engineering/quantisation-error). Hence, in terms of coding, we do not need to scale input samples to avoid overflow, shift the accumulator result to fit the [DAC](https://www.sciencedirect.com/topics/engineering/digital-to-analog-conversion) word size, scale the [filter coefficients](https://www.sciencedirect.com/topics/engineering/filter-coefficient), or apply Q-format arithmetic. A floating-point [DS processor](https://www.sciencedirect.com/topics/engineering/digital-signal-processor) with high speed and calculation precision facilitates a friendly environment to develop and implement DSP algorithms.

Analog Devices provides floating-point DSP families such as ADSP210xx and TigerSHARC. Texas Instruments offers a wide range of the floating-point DSP families, in which the TMS320C3x is the first generation, followed by the TMSC320C4x and TMS320C67x families. Since the first generation of a floating-point DS processor is less complicated than later generations but still has the common basic features, we review the first-generation architecture first.

Figure 9.14 shows the typical architecture of Texas Instruments’ TMS320C3x family of processors. We discuss some key features briefly. Further detail can be found in the TMS320C3x User’s Guide(Texas Instruments 1991), the TMS320C6x [CPU](https://www.sciencedirect.com/topics/engineering/central-processing-unit) and Instruction Set Reference Guide (Texas Instruments, 1998), and other studies (Dahnoun, 2000; Embree, 1995; Ifeachor and Jervis, 2002; Kehtaranavaz and Simsek, 2000; Sorensen and Chen, 1997; Van der Vegte, 2002). The TMS320C3x family consists of 32-bit single chip floating-point processors that support both integer and floating-point operations.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.14. The typical TMS320C3x floating-point DS processor.

The processor has a large memory space and is equipped with dual-access on-chip memories. A program cache is employed to enhance the execution of commonly used codes. Similar to the fixed-point processor, it uses the [Harvard architecture](https://www.sciencedirect.com/topics/engineering/harvard-architecture), where there are separate buses used for program and data so that instructions can be fetched at the same time that data are being accessed. There also exist memory buses and [data buses](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/channel-data-transmission) for direct-memory access (DMA) for concurrent I/O and CPU operations, and peripheral access such as [serial ports](https://www.sciencedirect.com/topics/engineering/serial-port), I/O ports, memory expansion, and an external clock.

The C3x CPU contains the floating-point/integer multiplier; an ALU, which is capable of operating both integer and floating-point arithmetic; a 32-bit barrel shifter; internal buses; a CPU register file; and dedicated [auxiliary](https://www.sciencedirect.com/topics/chemical-engineering/auxiliaries) register arithmetic units (ARAUs). The multiplier operates single-cycle multiplications on 24-bit integers and on 32-bit floating-point values. Using parallel instructions to perform a multiplication, an ALU will cost a single cycle, which means that a multiplication and an addition are equally fast. The ARAUs support addressing modes, in which some of them are specific to DSP such as [circular buffering](https://www.sciencedirect.com/topics/engineering/circular-buffering) and bit-reversal addressing (digital filtering and [FFT](https://www.sciencedirect.com/topics/engineering/fast-fourier-transform) operations). The CPU register file offers 28 registers, which can be operated on by the multiplier and ALU. The special functions of the registers include eight-extended 40-bit precision registers for maintaining accuracy of the floating-point results. Eight auxiliary registers can be used for addressing and for integer arithmetic. These registers provide internal temporary storage of internal variables instead of external memory storage, to allow performance of arithmetic between registers. In this way, program efficiency is greatly increased.

The prominent feature of C3x is its floating-point capability, allowing operation of numbers with a very large dynamic range. It offers implementation of the [DSP algorithm](https://www.sciencedirect.com/topics/engineering/digital-signal-processing-algorithm) without worrying about problems such as overflows and coefficient quantization. Three floating-point formats are supported. A short 16-bit floating-point format has 4 [exponent bits](https://www.sciencedirect.com/topics/engineering/exponent-bit), 1 sign bit, and 11 [mantissa](https://www.sciencedirect.com/topics/engineering/mantissa) bits. A 32-bit single precision format has 8 exponent bits, 1 sign bit, and 23 fraction bits. A 40-bit extended precision format contains 8 exponent bits, 1 sign bit, and 31 fraction bits. Although the formats are slightly different from the IEEE 754 standard, conversions are available between these formats.

The TMS320C30 offers high-speed performance with 60-nanosecond single-cycle instruction [execution time](https://www.sciencedirect.com/topics/engineering/execution-time), which is equivalent to 16.7 MIPS. For speech quality applications with an 8 kHz [sampling rate](https://www.sciencedirect.com/topics/engineering/sampling-rate), it can handle over 2,000 single-cycle instructions between two samples (125 microseconds). With instruction enhancements such as pipelines executing each instruction in a single cycle (four cycles required from fetch to execution by the instruction itself) and a multiple interrupt structure, this high-speed processor validates implementation of real-time applications in floating-point arithmetic.

Show more

Book

[2013, Digital Signal Processing (Second Edition)](https://www.sciencedirect.com/book/9780124158931/digital-signal-processing)

Li Tan, Jean Jiang

Chapter

[Hardware and Software for Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9780124158931000093)

Floating-Point Implementation Example

Real-time DSP implementation using a floating-point processor is easy to program. The overflow problem hardly ever occurs. Therefore, we do not need to consider scaling factors, as described in the last section. The code segment shown in Figure 9.28 demonstrates the simplicity of coding the floating-point [IIR filter](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/iir-filter) using the direct-form I structure.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.28. Sample C code for IIR filtering (floating-point implementation).

Book

[2013, Digital Signal Processing (Second Edition)](https://www.sciencedirect.com/book/9780124158931/digital-signal-processing)

Li Tan, Jean Jiang

Chapter

[Floating point](https://www.sciencedirect.com/science/article/pii/B978012819221400016X)

Abstract

This chapter introduces the [floating point](https://www.sciencedirect.com/topics/engineering/floating-point) operations that are equivalent to the operations provided by the vector floating point processor available on ARMv7 and earlier processors. An example is given of using them to implement the [sine function](https://www.sciencedirect.com/topics/engineering/sine-function), and the performance of the assembly function is compared to the performance of the sine function provided by the GNU C [standard library](https://www.sciencedirect.com/topics/social-sciences/information-library-standards).

Book

[2020, ARM 64-Bit Assembly Language](https://www.sciencedirect.com/book/9780128192214/arm-64-bit-assembly-language)

Larry D. Pyeatt, William Ughetta

Chapter

[Fixed-Point versus Floating-Point](https://www.sciencedirect.com/science/article/pii/B9780123744906000052)

5.3 Floating-Point Number Representation

Due to relatively limited dynamic ranges of fixed-point processors, when using such processors, one should be concerned with the scaling issue, or how big the numbers get in the manipulation of a signal. Scaling is not of concern when using floating-point processors, since the floating-point hardware provides a much wider dynamic range.

As an example, let us consider the C67x processor, which is the floating-point version of the TI family of TMS320C6000 DSP processors. There are two floating-point data representations on the C67x processor: single precision (SP) and double precision (DP). In the single-precision format, a value is expressed as (see [2])

(5.5)

where *s* denotes the sign bit (bit 31), *exp* denotes the [exponent bits](https://www.sciencedirect.com/topics/engineering/exponent-bit) (bits 23 through 30), and *frac*denotes the fractional or [mantissa](https://www.sciencedirect.com/topics/engineering/mantissa) bits (bits 0 through 22). (See Figure 5-6.)



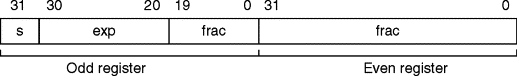
[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

Figure 5-6. C67x floating-point data representation.

Consequently, numbers as big as 3.4 × 1038 and as small as 1.175 × 10-38 can be processed. In the double-precision format, more fractional and exponent bits are used as indicated in

(5.6)

where the exponent bits are from bits 20 through 30, and the fractional bits are all the bits of a first word and bits 0 through 19 of a second word. (See Figure 5-7.) In this manner, numbers as big as 1.7×10308 and as small as 2.2×10-308 can be handled.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

Figure 5-7. C67x double-precision floating-point representation.

When one is using a floating-point processor, all the steps needed to perform floating-point arithmetic are done by the floating-point hardware. For example, consider adding two floating-point numbers represented by

(5.7)

The floating-point sum *c* has the following exponent and [fractional parts](https://www.sciencedirect.com/topics/engineering/fractional-part):

(5.8)

These parts are computed by the floating-point hardware. This shows that, though possible, it is inefficient to perform floating-point arithmetic on fixed-point processors, since all the operations involved, such as those in Equation (5.8), must be done in software.

Show more

Book

[2008, Digital Signal Processing System Design (Second Edition)](https://www.sciencedirect.com/book/9780123744906/digital-signal-processing-system-design)

Nasser Kehtarnavaz

Chapter

[Hardware and Software for Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9780124158931000093)

9.4 Fixed-Point and Floating-Point Formats

In order to process real-world data, we need to select an appropriate [DS processor](https://www.sciencedirect.com/topics/engineering/digital-signal-processor), as well as a [DSP algorithm](https://www.sciencedirect.com/topics/engineering/digital-signal-processing-algorithm) or algorithms for a certain application. Whether a DS processor uses a fixed- or floating-point method depends on how the processor’s [CPU](https://www.sciencedirect.com/topics/engineering/central-processing-unit) performs arithmetic. A fixed-point DS processor represents data in *2’s complement integer format* and manipulates data using integer arithmetic, while a floating-point processor represents number using a [mantissa](https://www.sciencedirect.com/topics/engineering/mantissa) (fractional part) and an exponent in addition to the integer format and operates data using floating-point arithmetic (discussed in Section 9.4.2).

Since the fixed-point DS processor operates using the integer format, which represents only a very narrow dynamic range of the [integer number](https://www.sciencedirect.com/topics/engineering/integer-number), a problem such as overflow of data manipulation may occur. Hence, we need to spend much more coding effort to deal with such a problem. As we shall see, we may use floating-point DS processors, which offer a wider dynamic range of data, so that coding becomes much easier. However, the floating-point DS processor contains more hardware units to handle the integer arithmetic and the floating-point arithmetic; hence it is more expensive and slower than fixed-point processors in terms of instruction cycles. It is usually a choice for prototyping or proof-of-concept development.

When it is time to make the DSP an application-specific integrated circuit (ASIC), a chip designed for a particular application, a dedicated hand-coded fixed-point implementation is likely the best choice in terms of performance and small [silica](https://www.sciencedirect.com/topics/materials-science/silicon-dioxide) area.

The formats used by DSP implementation can be classified as fixed or [floating point](https://www.sciencedirect.com/topics/engineering/floating-point).

**9.4.1 Fixed-Point Format**

We begin with 2’s complement representation. Considering a 3-bit 2’s complement, we can represent all the [decimal numbers](https://www.sciencedirect.com/topics/engineering/decimal-number) shown in Table 9.1.

Table 9.1. A 3-Bit 2’s Complement Number Representation

| **Decimal Number** | **Two’s Complement** |
| --- | --- |
| 3 | 011 |
| 2 | 010 |
| 1 | 001 |
| 0 | 000 |
| −1 | 111 |
| −2 | 110 |
| −3 | 101 |
| −4 | 100 |

Let us review the 2’s complement [number system](https://www.sciencedirect.com/topics/engineering/number-system) using Table 9.1. Converting a [decimal number](https://www.sciencedirect.com/topics/engineering/decimal-number) to its 2’s complement form requires following steps:

1.

Convert the magnitude in the decimal to its binary number using the required number of bits.

2.

If the decimal number is positive, its binary number is its 2’s complement representation; if the decimal number is negative, perform the 2’s complement operation, where we negate the binary number by changing the logic 1s to logic 0s and logic 0s to logic 1s and then add a logic 1 to the data. For example, a decimal number of 3 is converted to its 3-bit 2’s complement representation as 011; however, for converting a decimal number of −3, we first get a 3-bit binary number for the magnitude in decimal, that is, 011. Next, negating the binary number 011 yields the binary number 100. Finally, adding a [binary logic](https://www.sciencedirect.com/topics/engineering/binary-logic) 1 achieves the 3-bit 2’s complement representation of −3, that is, 100 +1=101, as shown in Table 9.1.

As we see, a 3-bit 2’s complement number system has a dynamic range from −4 to 3, which is very narrow. Since the basic DSP operations include multiplications and additions, results of operation can cause overflow problems. Let us examine multiplication in Example 9.1.

EXAMPLE 9.1

Given

a.

b.

operate each expression using 2’s complement.

**Solution**

a.

The 2’s complement of 00010 = 11110. Removing two extended sign bits 1 gives 110. The answer is 110 (−2), which is within the system.

b.

The 2’s complement of 00110 = 11010. Removing two extended sign bits leaves 010. Since the binary number 010 is 2, which is not (−6) as what we expect, overflow occurs; that is, the result of the multiplication (−6) is out of our dynamic range (−4 to 3).

Let us design a system treating all the decimal values as fractional numbers, so that we obtain the fractional binary 2’s complement system shown in Table 9.2.

Table 9.2. A 3-Bit 2’s Complement System Using Fractional Representation

| **Decimal Number** | **Decimal Fraction** | **Two’s Complement** |
| --- | --- | --- |
| 3 | 3/4 | 0.11 |
| 2 | 2/4 | 0.10 |
| 1 | 1/4 | 0.01 |
| 0 | 0 | 0.00 |
| −1 | −1/4 | 1.11 |
| −2 | −2/4 | 1.10 |
| −3 | −3/4 | 1.01 |
| −4 | −4/4 = −1 | 1.00 |

To become familiar with the fractional binary 2’s complement system, let us convert a positive fraction number and a negative fraction number − in decimals to their 2’s complements. Since

its 2’s complement is 011. Note that we did not mark the binary point for clarity. Again, since

its positive-number 2’s complement is 001. For the [negative number](https://www.sciencedirect.com/topics/engineering/negative-number), applying the 2’s complement to the binary number 001 leads to 110 + 1 = 111, as we see in Table 9.2. By adding the binary points, we obtain 0.01 and 1.11, respectively.

Now let us focus on the fractional binary 2’s complement system. The data are normalized to the fractional range from −1 to = . When we carry out multiplications with two fractions, the result should be a fraction, so that multiplication overflow can be prevented. Let us verify the multiplication (0.10) (1.01), which is the overflow case in Example 9.1. We first multiply two positive numbers:

The 2’s complement of 0.0110 = 1.1010.

The answer in decimal form should be

This number is correct, as we can verify from Table 9.2, that is, .

If we truncate the last two least significant bits to keep the 3-bit binary number, we have an approximate answer:

[Truncation error](https://www.sciencedirect.com/topics/engineering/truncation-error) occurs. The error should be bounded by . We can verify that

With such a scheme, we can avoid overflow due to multiplication but cannot prevent overflow due to addition. Consider the addition example

where the result 1.00 is a negative number. Adding two positive fractional numbers yields a negative number. Hence, overflow occurs.

We see that this signed fractional number scheme partially solves the overflow in multiplications. This fractional number format is called the signed Q-2 format, where there are 2 magnitude bits plus one sign bit. The overflow from addition will be tackled using a scaling method discussed in a later section.

Q-format number representation is the most common one used in fixed-point DSP implementation. It is defined in Figure 9.8.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.8. Q-15 (fixed-point) format.

As indicated in Figure 9.8, Q-15 means that the data are in a sign magnitude form in which there are 15 bits for magnitude and one bit for sign. Note that after the sign bit, the dot shown in Figure 9.8implies the binary point. The number is normalized to the fractional range from −1 to . The range is divided in to intervals, each with a size of . The most negative number is −1, while the most positive number is . Any result from multiplication is within the fractional range of −1 to 1. Let us study the following examples to become familiar with Q-format number representation.

EXAMPLE 9.2

Find the signed Q-15 representation for the decimal number 0.560123.

**Solution**

The conversion process is illustrated using Table 9.3. For a positive fractional number, we multiply the number by 2 if the product is larger than 1, carry bit 1 as a most significant bit (MSB), and copy the [fractional part](https://www.sciencedirect.com/topics/engineering/fractional-part) to the next line for the next multiplication by 2; if the product is less than 1, we carry bit 0 to MSB. The procedure continues to collect all 15 magnitude bits.

Table 9.3. Conversion Process of Q-15 Representation

| **Number** | **Product** | **Carry** |
| --- | --- | --- |
| 0.560123 × 2 | 1.120246 | 1 (MSB) |
| 0.120246 × 2 | 0.240492 | 0 |
| 0.240492 × 2 | 0.480984 | 0 |
| 0.480984 × 2 | 0.961968 | 0 |
| 0.961968 × 2 | 1.923936 | 1 |
| 0.923936 × 2 | 1.847872 | 1 |
| 0.847872 × 2 | 1.695744 | 1 |
| 0.695744 × 2 | 1.391488 | 1 |
| 0.391488 × 2 | 0.782976 | 0 |
| 0.782976 × 2 | 1.565952 | 1 |
| 0.565952 × 2 | 1.131904 | 1 |
| 0.131904 × 2 | 0.263808 | 0 |
| 0.263808 × 2 | 0.527616 | 0 |
| 0.527616 × 2 | 1.055232 | 1 |
| 0.055232 × 2 | 0.110464 | 0 (LSB) |

MSB, most significant bit; LSB, least-significant bit.

We yield the Q-15 format representation as

Since we only use 16 bits to represent the number, we may lose accuracy after conversion. Like quantization, truncation error is introduced. However, this error should be less than the interval size, in this case, . We shall verify this in Example 9.5. An alternative method of conversion is to convert a fraction, let’s say to Q-2 format, multiply it by , and then convert the truncated integer to its binary, that is,

In this way, it follows that

Converting 18,354 to its binary representation will achieve the same answer. The next example illustrates the signed Q-15 representation for a negative number.

EXAMPLE 9.3

Find the signed Q-15 representation for the decimal number −0.160123.

**Solution**

Converting the Q-15 format for the corresponding positive number with the same magnitude using the procedure described in Example 9.2, we have

Then after applying 2’s complement, the Q-15 format becomes

*Alternative method*: Since , converting the truncated number −5,246 to its 16-bit 2’s complement yields 1110101110000010.

EXAMPLE 9.4

Convert the Q-15 signed number 1.110101110000010 to the decimal number.

**Solution**

Since the number is negative, applying the 2’s complement yields

Then the decimal number is

EXAMPLE 9.5

Convert the Q-15 signed number 0.100011110110010 to the decimal number.

**Solution**

The decimal number is

As we know, the truncation error in Example 9.2 is less than . We verify that the truncation error is bounded by

Note that the larger the number of bits used, the smaller the truncation error that may accompany it.

Examples 9.6 and 9.7 are devoted to illustrating data manipulations in the Q-15 format.

EXAMPLE 9.6

Add the two numbers in Examples 9.4 and 9.5 in Q-15 format.

**Solution**

Binary addition is carried out as follows:

Then the result is

This number in decimal form is

EXAMPLE 9.7

This is a simple illustration of fixed-point multiplication.

Determine the fixed-point multiplication of 0.25 and 0.5 in Q-3 fixed-point 2’s complement format.

**Solution**

Since 0.25 = 0.010 and 0.5 = 0.100, we carry out [binary multiplication](https://www.sciencedirect.com/topics/engineering/binary-multiplication) as follows:

Truncating the least significant bits to convert the result to Q-3 format, we have

Note that . We can also verify that 0.25 0.5 = 0.125.

The Q-format number representation is a better choice than the 2’s complement integer representation, it can prevent multiplication overflow. But we need to be concerned with the following problems.

1.

When converting a decimal number to its Q- format, where denotes the number of magnitude bits, we may lose accuracy due to the truncation error, which is bounded by the size of the interval, that is, .

2.

Addition and subtraction may cause overflow, where adding two positive numbers leads to a negative number, or adding two negative number yields a positive number; similarly, subtracting a positive number from a negative number gives a positive number, while subtracting a negative number from a positive number results in a negative number.

3.

Multiplying two numbers in Q-15 format will lead to a Q-30 format, which has 31 bits in total. As in Example 9.7, the multiplication of Q-3 yields a Q-6 format, that is, 6 magnitude bits and a sign bit. In practice, it is common for a DS processor to hold the multiplication result using a double word size such as MAC operation, as shown in Figure 9.9 for multiplying two numbers in Q-15 format. In Q-30 format, there is one sign-extended bit. We may get rid of it by shifting left by one bit to obtain Q-31 format and maintaining the Q-31 format for each MAC operation.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.9. Sign bit extended Q-30 format.

Sometimes, the number in Q-31 format needs to be converted to Q-15; for example, the 32-bit data in the accumulator needs to be sent for 16-bit digital-to-analog conversion (DAC), where the upper most-significant 16 bits in the Q-30 format must be used to maintain accuracy. We can shift the number in Q-30 to the right by 15 bits or shift the Q-31 number to the right by 16 bits. The useful result is stored in the lower 16-bit memory location. Note that after truncation, the maximum error is bounded by the interval size of , which satisfies most applications. In using the Q-format in the fixed-point DS processor, it is costly to maintain the accuracy of data manipulation.

4.

Underflow can happen when the result of multiplication is too small to be represented in the Q-format. As an example, in a Q-2 system shown in Table 9.2, multiplying 0.01 0.01 leads to 0.0001. To keep the result in Q-2, we truncate the last two bits of 0.0001 to achieve 0.00, which is zero. Hence, underflow occurs.

**9.4.2 Floating-Point Format**

To increase the dynamic range of number representation, a floating-point format, which is similar to scientific notation, is used. The general format for floating-point number representation is given by

(9.1)

where is the mantissa, or fractional part in Q format, and is the exponent. The mantissa and exponent are signed numbers. If we assign 12 bits for the mantissa and 4 bits for the exponent, the format looks like Figure 9.10.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.10. Floating-point format.

Since the 12-bit mantissa is limited to between −1 to +1, the number of bits assigned to the exponent controls the dynamic range. The bigger the number of bits designated to the exponent, the larger the dynamic range. The number of bits for the mantissa defines the interval in the normalized range; as shown in Figure 9.10, the interval size is in the normalized range, which is smaller than the Q-15 format. However, when more mantissa bits are used, there will be a smaller interval size. Using the format in Figure 9.10, we can determine the most negative and most positive numbers as

The smallest positive number is given by

As we can see, the exponent acts like a scale factor to increase the dynamic range of the number representation. We study the floating-point format in the following example.

EXAMPLE 9.8

Convert each of the following decimal numbers to a floating-point number using the format specified in Figure 9.10.

a.

0.1601230

b.

−20.430527

**Solution**

a. We first scale the number 0.1601230 to with an exponent of −2 (other choices could be 0 or −1) to get . Using 2’s complement, we have . Now we convert the value using the Q-11 format to get 010100011111. Cascading the [exponent bits](https://www.sciencedirect.com/topics/engineering/exponent-bit) and the mantissa bits yields

b. Since , we can convert it into the fractional part and exponent part as . Note that this conversion is not particularly unique; the forms and are still valid choices. Let us keep what we have now. Therefore, the exponent bits should be 0101.

Converting the number 0.638454 using the Q-11 format gives

Using 2’s complement, we obtain the representation for the decimal number −0.6438454 as

Cascading the exponent bits and mantissa bits, we achieve

Floating-point arithmetic is more complicated. We must obey the rules for manipulating two floating-point numbers. For arithmetic addition, with two floating point numbers given as

the floating-point sum is performed as follows:

For multiplication, given two properly normalized floating-point numbers

where and , the calculation can be performed as follows:

That is, the mantissas are multiplied while the exponents are added:

Examples 9.9 and 9.10 serve to illustrate manipulators.

EXAMPLE 9.9

Add the two floating point numbers obtained in Example 9.8:

**Solution**

Before addition, we change the first number so it has the same exponent as the second number, that is,

Then we add the two mantissa numbers:

The floating number is

We can verify the result by the following:

EXAMPLE 9.10

Multiply the two floating-point numbers obtained in Example 9.8:

**Solution**

From the results in Example 9.8, we have the bit patterns for these two numbers as

Adding two exponents in 2’s complement form leads to

which is +3, as we expected, since in the decimal domain (−2) + 5 = 3. As previously shown when introducing the multiplication rule, when multiplying two mantissas, we need to apply their corresponding positive values. If the sign for the final value is negative, then we convert it to its 2’s complement form. In our example, is a positive mantissa. However, is a negative mantissa, since the MSB is 1. To perform multiplication, we use 2’s complement to convert to its positive value, 010100011011, and note that the multiplication result is negative. We multiply two positive mantissas and truncate the result to 12 bits to give

Now we need to add a negative sign to the multiplication result with the 2’s complement operation. Taking the 2’s complement, we have

Hence, the product is achieved by cascading the 4-bit exponent and 12-bit mantissa as

Converting this number back to the decimal number, we verify the result to be

Next, we examine overflow and underflow in the floating-point number system.

**Overflow**

During an operation, overflow will occur when a number is too large to be represented in the floating-point number system. Adding two mantissa numbers may lead to a number larger than 1 or less than −1; and multiplying two numbers causes the addition of their two exponents so that the sum of the two exponents could overflow. Consider the following overflow cases.

**Case 1.** Add the following two floating-point numbers:

Note that the two exponents are the same and they are the biggest positive number in 4-bit 2’s complement representation. We add two positive mantissa numbers as

The result for adding mantissa numbers is negative. Hence the overflow occurs.

**Case 2:** Multiply the following two numbers:

Adding the two positive exponents gives

Multiplying the two mantissa numbers gives

**Underflow**

As we discussed before, underflow will occur when a number is too small to be represented in the number system. Let us divide the following two floating-point numbers:

First, subtracting the two exponents leads to

Then, dividing two mantissa numbers, it follows that

However, in this case, the expected resulting exponent is −14 in decimal, which is too small to be presented in the 4-bit 2’s complement system. Hence the underflow occurs.

Now that we understand the basic principles of the floating-point formats, we can next examine two floating-point formats of the Institute of Electrical and Electronics Engineers (IEEE).

**9.4.3 IEEE Floating-Point Formats**

**Single Precision Format**

IEEE floating-point formats are widely used in many modern DS processors. There are two types of IEEE floating-point formats (IEEE 754 standard). One is the IEEE single precision format, and the other is the IEEE double precision format. The single precision format is described in Figure 9.11.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.11. IEEE single precision floating-point format.

The format of IEEE single precision floating-point standard representation requires 23 fraction bits , 8 exponent bits , and 1 sign bit , with a total of 32 bits for each word. is the mantissa in 2’s complement positive binary fraction represented from bit 0 to bit 22. The mantissa is within the normalized range limits between +1 and +2. The sign bit is employed to indicate the sign of the number, where when the number is negative, and when the number is positive. The exponent is in excess 127 form. The value of 127 is the offset from the 8-bit exponent range from 0 to 255, so that E-127 will have a range from −127 to +128. The formula shown in Figure 9.11 can be applied to convert the IEEE 754 standard (single precision) to the decimal number. The following simple examples also illustrate this conversion:

Let us look at Example 9.11 for more explanation.

EXAMPLE 9.11

Convert the following number in IEEE single precision format to decimal format:

**Solution**

From the bit pattern in Figure 9.11, we can identify the sign bit, exponent, and fractional as

Then, applying the conversion formula leads to

In conclusion, the value  represented by the word can be determined based on the following rules, including all the exceptional cases:

•

If and is nonzero, then ("Not a number").

•

If , is zero, and is 1, then .

•

If , is zero, and is 0, then .

•

If , then , where represents the binary number created by prefixing with an implicit leading 1 and a binary point.

•

If and is nonzero, then . This is an "unnormalized" value.

•

If , is zero, and is 1, then .

•

If , is zero, and is 0, then .

Typical and exceptional examples are shown as follows:

000000000 00000000000000000000000 = 0

100000000 00000000000000000000000 = −0

011111111 00000000000000000000000 = Infinity

111111111 00000000000000000000000 = −Infinity

011111111 00000100000000000000000 = NaN

111111111 00100010001001010101010 = NaN

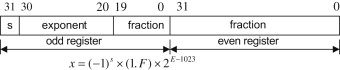
000000001 00000000000000000000000 =

000000000 10000000000000000000000 =

000000000 00000000000000000000001 =

**Double Precision Format**

The IEEE double precision format is described in Figure 9.12.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.12. IEEE double precision floating-point format.

The IEEE double precision floating-point standard representation requires a 64-bit word, which may be numbered from 0 to 63, left to right. The first bit is the sign bit , the next eleven bits are the exponent bits , and the final 52 bits are the fraction bits . The IEEE floating-point format in double precision significantly increases the dynamic range of number representation since there are eleven exponent bits; the double-precision format also reduces the interval size in the mantissa normalized range of +1 to +2, since there are 52 mantissa bits as compare with the single precision case of 23 bits. Applying the conversion formula shown in Figure 9.12 is similar to the single precision case.

EXAMPLE 9.12

Convert the following number in IEEE double precision format to the decimal format:

**Solution**

Using the bit pattern in Figure 9.12, we have

Then, applying the double precision formula yields

For the purpose of completeness, rules for determining the value  represented by the double-precision word are listed as follows:

•

If and is nonzero, then ("Not a number").

•

If , is zero, and is 1, then .

•

If , is zero, and is 0, then .

•

If , then , where " " is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.

•

If and is nonzero, then . This is an "unnormalized" value.

•

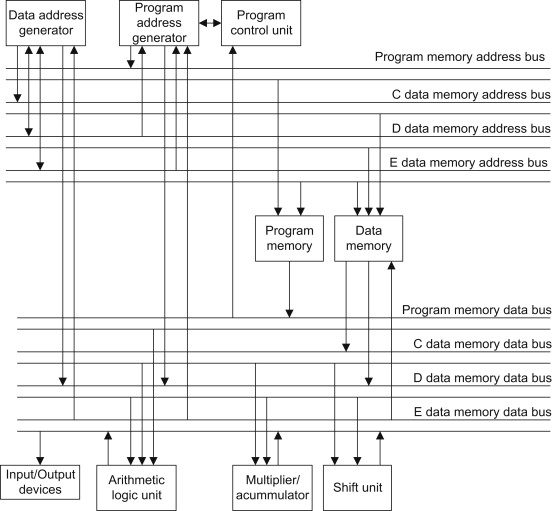
If , is zero, and is 1, then .

•

If , is zero, and is 0, then .

**9.4.5 Fixed-Point Digital Signal Processors**

Analog Devices, Texas Instruments, and Motorola all manufacture fixed-point DS processors. Analog Devices offers a fixed-point DSP family such as the ADSP21xx. Texas Instruments provides various generations of fixed-point DS processors based on historical development, architecture features, and computational performance. Some of the most common ones are the TMS320C1x (first generation), TMS320C2x, TMS320C5x, and TMS320C62x. Motorola manufactures a variety of fixed-point processors, such as the DSP5600x family. The new families of fixed-point DS processors are expected to continue to grow. Since they share some basic common features such as program memory and data memory with associated address buses, arithmetic [logic units](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/arithmetic-and-logic-units) (ALUs), program [control units](https://www.sciencedirect.com/topics/engineering/control-unit), MACs, shift units, and [address generators](https://www.sciencedirect.com/topics/engineering/address-generator), here we focus on an overview of the TMS320C54x processor. The typical TMS320C54x fixed-point DSP architecture appears in Figure 9.13.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.13. Basic architecture of the TMSC320C54x family.

The fixed-point TMS320C50 families supporting 16-bit data have on-chip program memory and data memory in various sizes and configurations. They include data [RAM](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/random-access-memory) (random access memory) and program ROM (read-only memory) used for program code, instruction, and data. Four [data buses](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/channel-data-transmission) and four address buses are accommodated to work with the data memory and program memory. The program memory address bus and program memory [data bus](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/channel-data-transmission) are responsible for fetching program instructions. As shown in Figure 9.13, the C and D data memory address buses and the C and D data memory data buses deal with fetching data from the data memory while the E data memory address bus and E data memory data bus are dedicated to moving data into data memory. In addition, the E memory data bus can access the I/O devices.

Computational units consist of an ALU, a MAC, and a shift unit. For the TMS320C54x family, the ALU can fetch data from the C, D, and program memory data buses and access the E memory data bus. It has two independent 40-bit accumulators, which are able to operate 40-bit addition. The multiplier, which can fetch data from the C and D memory data buses and write data via the E data memory data bus, is capable of operating 17-bit  17-bit multiplications. The 40-bit shifter has the same capability of bus access as the MAC, allowing all possible shifts for scaling and fractional arithmetic such as those we have discussed for the Q-format.

The program [control unit](https://www.sciencedirect.com/topics/engineering/control-unit) fetches instructions via the program memory data bus. Again, in order to speed up memory access, there are two address generators available: one responsible for program addresses and one for data addresses.

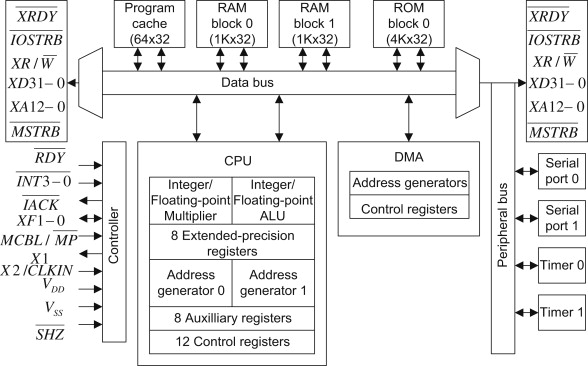
Advanced [Harvard architecture](https://www.sciencedirect.com/topics/engineering/harvard-architecture) is employed, where several instructions operate at the same time for given a given single instruction cycle. Processing performance offers 40 MIPS (million instruction sets per second). To further explore this subject, the reader is referred to Dahnoun (2000), Embree (1995), Ifeachor and Jervis (2002), and Van der Vegte (2002), as well as the TI website (www.ti.com).

**9.4.6 Floating-Point Processors**

Floating-point DS processors perform DSP operations using floating-point arithmetic, as we discussed before. The advantages of using the floating-point processor include getting rid of finite word length effects such as overflows, round-off errors, truncation errors, and coefficient [quantization error](https://www.sciencedirect.com/topics/engineering/quantisation-error). Hence, in terms of coding, we do not need to scale input samples to avoid overflow, shift the accumulator result to fit the [DAC](https://www.sciencedirect.com/topics/engineering/digital-to-analog-conversion) word size, scale the [filter coefficients](https://www.sciencedirect.com/topics/engineering/filter-coefficient), or apply Q-format arithmetic. A floating-point DS processor with high speed and calculation precision facilitates a friendly environment to develop and implement DSP algorithms.

Analog Devices provides floating-point DSP families such as ADSP210xx and TigerSHARC. Texas Instruments offers a wide range of the floating-point DSP families, in which the TMS320C3x is the first generation, followed by the TMSC320C4x and TMS320C67x families. Since the first generation of a floating-point DS processor is less complicated than later generations but still has the common basic features, we review the first-generation architecture first.

Figure 9.14 shows the typical architecture of Texas Instruments’ TMS320C3x family of processors. We discuss some key features briefly. Further detail can be found in the TMS320C3x User’s Guide(Texas Instruments 1991), the TMS320C6x CPU and Instruction Set Reference Guide (Texas Instruments, 1998), and other studies (Dahnoun, 2000; Embree, 1995; Ifeachor and Jervis, 2002; Kehtaranavaz and Simsek, 2000; Sorensen and Chen, 1997; Van der Vegte, 2002). The TMS320C3x family consists of 32-bit single chip floating-point processors that support both integer and floating-point operations.



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

FIGURE 9.14. The typical TMS320C3x floating-point DS processor.

The processor has a large memory space and is equipped with dual-access on-chip memories. A program cache is employed to enhance the execution of commonly used codes. Similar to the fixed-point processor, it uses the Harvard architecture, where there are separate buses used for program and data so that instructions can be fetched at the same time that data are being accessed. There also exist memory buses and data buses for direct-memory access (DMA) for concurrent I/O and CPU operations, and peripheral access such as [serial ports](https://www.sciencedirect.com/topics/engineering/serial-port), I/O ports, memory expansion, and an external clock.

The C3x CPU contains the floating-point/integer multiplier; an ALU, which is capable of operating both integer and floating-point arithmetic; a 32-bit barrel shifter; internal buses; a CPU register file; and dedicated [auxiliary](https://www.sciencedirect.com/topics/chemical-engineering/auxiliaries) register arithmetic units (ARAUs). The multiplier operates single-cycle multiplications on 24-bit integers and on 32-bit floating-point values. Using parallel instructions to perform a multiplication, an ALU will cost a single cycle, which means that a multiplication and an addition are equally fast. The ARAUs support addressing modes, in which some of them are specific to DSP such as [circular buffering](https://www.sciencedirect.com/topics/engineering/circular-buffering) and bit-reversal addressing (digital filtering and [FFT](https://www.sciencedirect.com/topics/engineering/fast-fourier-transform) operations). The CPU register file offers 28 registers, which can be operated on by the multiplier and ALU. The special functions of the registers include eight-extended 40-bit precision registers for maintaining accuracy of the floating-point results. Eight auxiliary registers can be used for addressing and for integer arithmetic. These registers provide internal temporary storage of internal variables instead of external memory storage, to allow performance of arithmetic between registers. In this way, program efficiency is greatly increased.

The prominent feature of C3x is its floating-point capability, allowing operation of numbers with a very large dynamic range. It offers implementation of the DSP algorithm without worrying about problems such as overflows and coefficient quantization. Three floating-point formats are supported. A short 16-bit floating-point format has 4 exponent bits, 1 sign bit, and 11 mantissa bits. A 32-bit single precision format has 8 exponent bits, 1 sign bit, and 23 fraction bits. A 40-bit extended precision format contains 8 exponent bits, 1 sign bit, and 31 fraction bits. Although the formats are slightly different from the IEEE 754 standard, conversions are available between these formats.

The TMS320C30 offers high-speed performance with 60-nanosecond single-cycle instruction [execution time](https://www.sciencedirect.com/topics/engineering/execution-time), which is equivalent to 16.7 MIPS. For speech quality applications with an 8 kHz [sampling rate](https://www.sciencedirect.com/topics/engineering/sampling-rate), it can handle over 2,000 single-cycle instructions between two samples (125 microseconds). With instruction enhancements such as pipelines executing each instruction in a single cycle (four cycles required from fetch to execution by the instruction itself) and a multiple interrupt structure, this high-speed processor validates implementation of real-time applications in floating-point arithmetic.

Show more

Book

[2013, Digital Signal Processing (Second Edition)](https://www.sciencedirect.com/book/9780124158931/digital-signal-processing)

Li Tan, Jean Jiang

Chapter

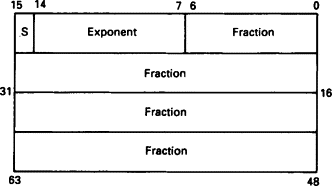
[Computers and their application](https://www.sciencedirect.com/science/article/pii/B9780750611954500087)

4.7.15 Fixed and floating-point arithmetic hardware

As far as arithmetic instructions go, simpler CPUs only contain add and subtract instructions, operating on single-word operands. Multiplication, of both fixed and floatingpoint numbers, is then accomplished by software subroutines, i.e. standard programs which perform multiplication or division by repetitive use of the add or subtract instructions, which can be invoked by a programmer who requires to perform a multiplication or division operation.

By providing extra hardware to perform fixed-point multiply and divide, which also usually implements multiple place-shift operations, a very substantial improvement in the speed of multiply and divide operations is obtained. With the hardware techniques used to implement most modern CPUs, however, these instructions are wired in as part of the standard set.

Floating-point format (Figure 4.5) provides greater range and precision than single-word fixed-point format. In floatingpoint representation, numbers are stored as a fraction times 2*n* where *n* can be positive or negative. The fraction (or mantissa) and exponent are what is stored, usually in two words for single-precision floating-point format or four words for [double precision](https://www.sciencedirect.com/topics/computer-science/double-precision).



[Sign in to download full-size image](https://www.sciencedirect.com/user/login?returnURL=https%3A%2F%2Fwww.sciencedirect.com%2Ftopics%2Fengineering%2Ffloating-point-processor)

Figure 4.5. 32-bit double floating-point format

Hardware to perform add, subtract, multiply and divide operations is sometimes implemented as a floating-point processor, an independent unit with its own registers to which floating-point instructions are passed. The floating-point processor (sometimes called co-processor) can then access the operands, perform the required [arithmetic operation](https://www.sciencedirect.com/topics/computer-science/arithmetic-operation) and signal the CPU, which has meanwhile been free to continue with its own processing until the result is available.

An independent floating-point processor clearly provides the fastest execution of these instructions, but even without that, implementing them within the normal instruction set of the CPU, using its addressing techniques to access operands in memory, provides a significant improvement over software subroutines. The inclusion of the [FPP](https://www.sciencedirect.com/topics/computer-science/false-positive-problem) into ‘standard’ CPUs is becoming almost standard.

Book

[1994, Mechanical Engineer's Reference Book (Twelfth Edition)](https://www.sciencedirect.com/book/9780750611954/mechanical-engineers-reference-book)

Ian Robertson

Chapter

[Floating-Point Representation, Algorithms, and Implementations](https://www.sciencedirect.com/science/article/pii/B9781558607989500105)

8.3 IEEE Standard 754

As we have seen, there are many parameters that define a floating-point representation system. This resulted in a variety of floating-point processors with different representations, producing different results to the execution of the same program. In some cases, because of anomalies, the results might be very different. To avoid this, the IEEE Floating-point Standard 754 was developed. It is claimed that this standard

•

minimizes anomalies

•

enhances portability

•

enhances numerical quality

•

allows different implementations

We now describe the main components of the IEEE Standard 754, which is used today by most floating-point processors.11

**8.3.1 Representation and Formats**

The two parts of the representation are as follows:

First, the significand is in sign-and-magnitude representation. Consequently, it is represented by two components:

•

Sign S. One bit. S = 1 if negative.

•

*Magnitude* (also called the significand). Represented in [radix 2](https://www.sciencedirect.com/topics/computer-science/radix-2) with one integer bit. That is, the normalized significand is represented by

8.44

where *F* of *f* bits (depending on the format) is called the fraction and the most-significant 1 is the *hidden bit*. The range of the (normalized) significand is

8.45

Second, the exponent is base 2 and in biased representation. The number of bits of the exponent field is e, depending on the format. The representation is biased with bias *B* = 2e−1−1.

The three components are packed into one word, in which the order of the fields is *S*, *E*, *F*.12 This order makes comparisons simpler.

The value zero, denormals, and the special values NAN and infinities are represented as follows:

•

The representation of floating-point zero is *E* = 0 and *F* = 0. The sign *S* differentiates between positive and negative zero. Because of this representation and the hidden bit, the value 1.0 × 2−Bis not represented.

•

The representation E = 0 and F ≠ 0 is used for denormals; in this case13 the floating-point value represented is *v*=(-1)s2-(B-1)(0.*F*).

•

The maximum exponent representation (E = 2e − 1 = 2*B* + 1) is used to represent not-a-number (NAN) for *F* ≠ 0 and plus and minus infinity for *F* = 0.

The system has two formats: basic and extended. Moreover, the basic format allows representation in single and [double precision](https://www.sciencedirect.com/topics/computer-science/double-precision). We now describe these formats.In each case we give the three components with the number of bits in parentheses. We call v the value represented.

1.

Basic: single (32 bits) and double (64 bits)

•

Single: *S*(1), *E*(8), *F*(23)

(a)

Ifl ≤ *E* ≤ 254, then *v* = (−1)S2E-127(1.*F*) (normalized fp number).

(b)

If *E* = 255 and *F* ≠ 0, then *v* = *NAN* (not a number).

(c)

If *E* = 255 and *F* = 0, then *v* = (−l)s ∞(plus and minus infinity).

(d)

If *E* = 0 and *F*≠0,then *v* = (−1)S2E-126(0.*F*) (denormal, gradual underflow).

(e)

If *E* = 0 and *F* = 0, then *v* = (−1)S 0 (positive and negative zero).

•

Double: *S*(1), *E*(11), *F*(52)

-

Similar representation to single, replacing 255 by 2047, and so on.

2.

Extended: single (at least 43 bits = *S*(1), *E*(11), *F*(31)) and double (at least 79 bits = *S*(1), *E*(15), *F*(63)).

**8.3.2 Rounding**

[Rounding modes](https://www.sciencedirect.com/topics/computer-science/rounding-mode) are:

•

Default: Round to nearest, to even when tie

•

Directed: Round toward plus infinity; Round toward minus infinity; and Round toward 0 (truncate)

**8.3.3 Operations**

Operations include:

•

Numerical: Add, Sub, Mult, Div, Square root, Rem

•

Conversions: Floating to integer; Binary to decimal (integer); Binary to decimal (floating)

•

Miscellaneous: Change formats; Compare and set condition code

**8.3.4 Exceptions**

The IEEE standard defines the following five exceptions. By default these exceptions set a flag and the computation continues. The implementation can include a trap handler for each exception that, when enabled, is called when an exception occurs.

•

Overflow (when rounded value is too large to be represented). Result is set to ±infinity.

•

Underflow (when rounded value is too small to be represented).

•

Division by zero.

•

Inexact result (result is not an exact floating-point number). Infinite precision result different from floating-point number.

•

Invalid. This flag is set when a NAN result is produced.

Show more

Book

[2004, Digital Arithmetic](https://www.sciencedirect.com/book/9781558607989/digital-arithmetic)

Miloš D. Ercegovac, Tomás Lang

Chapter

[Hardware and Software for Digital Signal Processors](https://www.sciencedirect.com/science/article/pii/B9780124158931000093)

9.6.4 Sample C Programs

**Floating-Point Implementation Example**

Real-time DSP implementation using a floating-point processor is easy to program. The overflow problem hardly ever occurs. Therefore, we do not need to consider scaling factors, as described in the last section. The code segment shown in Figure 9.28 demonstrates the simplicity of coding the floating-point [IIR filter](https://www.sciencedirect.com/topics/earth-and-planetary-sciences/iir-filter) using the direct-form I structure.